**SYLLABUS**

## 1. Information regarding the programme

| | |
|---|---|
| 1.1 Higher education institution | **Babeş-Bolyai University of Cluj-Napoca** |
| 1.2 Faculty | **Faculty of Mathematics and Computer Science** |
| 1.3 Departament | **Departament of Computer Science** |
| 1.4 Field of study | **Computer Science** |
| 1.5 Study Cycle | **Bachelor** |
| 1.6 Study Cycle / Qualification | **Computer Science** |

## 2. Information regarding the discipline

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.1 Name of the discipline | **Fundamentals of Programming** | | | | | | |
| 2.2 Course coordinator | **Lect. PhD. Molnar Arthur** | | | | | | |
| 2.3 Seminar coordinator | **Lect. PhD. Molnar Arthur** | | | | | | |
| 2.4 Year of study | **1** | 2.5 Semester | **1** | 2.6. Type of evaluation | **E** | 2.7. Type of discipline | **Compulsory** |

**3. Total estimated time** (hours/semester of didactic activities)

| | | | | | |
|---|---|---|---|---|---|
| 3.1 Hours per week | 6 | Of which: 3.2 course | 2 | 3.3 seminar/laboratory | 2 sem 2 lab |
| 3.4 Total hours in the curriculum | 84 | Of which: 3.5 course | 28 | 3.6 seminar/laboratory | 56 |
| Time allotment: | | | | | hours |
| Learning using manual, course support, bibliography, course notes | | | | | 14 |
| Additional documentation (in libraries, on electronic platforms, field documentation) | | | | | 12 |
| Preparation for seminars/labs, homework, papers, portfolios and essays | | | | | 14 |
| Tutorship | | | | | 8 |
| Evaluations | | | | | 18 |
| Other activities: ..... | | | | | |

| | |
|---|---|
| 3.7 Total individual study hours | 66 |
| 3.8 Total hours per semester | 150 |
| 3.9 Number of ECTS credits | 6 |

**4. Prerequisites** (if necessary)

| | |
|---|---|
| 4.1 curriculum | - |
| 4.2 competencies | - |

**5. Conditions** (if necessary)

| | |
|---|---|
| 5.1 For the course | Class room with projector |
| 5.2 For the seminar/lab activities | • Laboratory with computers;<br>• Python programming language and environment |

**6. Specific competencies acquired**

| | |
|---|---|
| **Professional competencies** | • C1.1 Description of programming paradigms and of language specific mechanisms, as well as identification of syntactic and semantic differences.<br>• C1.2 Explanation of existing software applications, on different levels of abstraction (architecture, classes, methods) using adequate basic knowledge.<br>• C1.3 Elaboration of adequate source code and testing of components in a given programming language, based on given specifications.<br>• C1.4 Testing applications based on testing plans.<br>• C1.5 Developing units of programs and corresponding documentation. |
| **Transversal competencies** | • CT1 Application of efficient and rigorous working rules, manifest responsible attitudes towards the scientific and didactic fields, respecting professional and ethical principles.<br>• CT2 Use of efficient methods and techniques for learning, information, research and development of abilities for knowledge exploitation, for adapting to the needs of a dynamic society and for communication in a widely used foreign language. |

**7. Objectives of the discipline** (outcome of the acquired competencies)

| | |
|---|---|
| 7.1 General objective of the discipline | To know the basic concepts of software engineering (design, implementation and maintenance |
| 7.2 Specific objectives of the discipline | • To know the key concepts of programming<br>• To know the basic concepts of software engineering (design, implementation and maintenance of software systems).<br>• To gain understanding of basic software tools used in development and testing.<br>• To learn Python programming language, and to get used to Python programming, running, testing, and debugging programs.<br>• To acquire and improve their individual programming style. |

**8. Content**

| **8.1 Lecture** | Teaching methods | Remarks |
|---|---|---|
| **1. Introduction to software development processes**<br>• What is programming: algorithm, program, basic elements of the Python language, Python interpreter, basic roles in software engineering<br>• How to write programs: problem statement, requirements, feature driven development process<br>• Example: calculator, iteration modelling | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples<br>• Didactical demonstration | |
| **2. Procedural programming**<br>• Compound types: list, tuple, dictionary<br>• Functions: test cases, definition, variable scope, calling, parameter passing<br>• Test-driven development (TDD) steps, refactoring | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples<br>• Didactical demonstration | |
| **3. Modular programming**<br>• What is a module: Python module definition, variable scope in a module, packages, standard module libraries, deployment | • Interactive exposure<br>• Explanation | |

| | | |
|---|---|---|
| • How to organize source code: responsibilities, single responsibility principle, separation of concerns, dependency, coupling, cohesion<br>• Common layers in an information system - logical architecture<br>• Eclipse + PyDev | • Conversation<br>• Examples<br>• Didactical demonstration | |
| **4. User defined types**<br>• How to define new data types: encapsulation, information hiding (data hiding in Python), guidelines, abstract data types | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples<br>• Didactical demonstration | |
| **5. Design guidelines**<br>• Problem statement: a program for managing information (CRUD operations)<br>• Layered architecture: UI layer, application layer, domain layer, infrastructure layer<br>• GRASP patterns<br>• Example of application development: entity, validator, repository, controller<br>• Principles: information expert, low coupling, high cohesion, protected variation, single responsibility, dependency injection | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples<br>• Didactical demonstration | |
| **6. Object based programming**<br>• Objects and classes: classes, objects, fields, methods, special class methods (operator overloading), Python scope and namespace<br>• UML Diagrams: class diagrams, relationships, associations, invariants<br>• Inheritance: UML generalization, code reuse, overriding, inheritance in Python<br>• Exceptions<br>• Example: working with files in Python, repository implementations using files | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples<br>• Didactical demonstration | |
| **7. Program design**<br>• Top down and bottom up strategies: top down design, bottom up design, bottom up programming style, mixed approach<br>• Organizing the UI<br>• Class invariants | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples<br>• Didactical demonstration | |
| **8. Program testing and inspection**<br>• Testing methods: exhaustive testing, black box testing, white box testing<br>• Testing levels: unit testing, integration testing<br>• Automated testing, TDD<br>• Program inspection: coding style, refactoring | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples<br>• Didactical demonstration | |
| **9. Recursion**<br>• Notion of recursion<br>• Direct and indirect recursion<br>• Examples | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples | |

| | | |
|---|---|---|
| | • Didactical demonstration | |
| **10. Algorithm complexity**<br>• Empiric analysis and asymptotic analysis<br>• Asymptotic notation: big-o, little-o, big-omega, little-omega, theta; properties<br>• Examples of magnitude orders<br>• Comparison of algorithms from an efficiency point of view<br>• Structural complexity | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples<br>• Didactical demonstration | |
| **11. Searching. Sorting.**<br>• Specification of the searching/sorting problem<br>• Search methods: sequential, binary.<br>• Sort methods: BubbleSort, SelectionSort, InsertionSort, QuickSort, MergeSort<br>• Complexity of searching/sorting algorithms | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples<br>• Didactical demonstration | |
| **12. Problem solving methods (I)**<br>• General presentation of the Greedy and Backtracking methods<br>• Algorithms and complexity<br>Examples | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples<br>• Didactical demonstration | |
| **13. Problem solving methods (II)**<br>• General presentation of the Divide & Conquer and Dynamic Programming methods<br>• Algorithms and complexity<br>• Examples | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples<br>• Didactical demonstration | |
| **14. Revision**<br>• Revision of most important topics covered by the course<br>• Exam guide | • Interactive exposure<br>• Explanation<br>• Conversation<br>• Examples<br>• Didactical demonstration | |

**Bibliography**


1. Kent Beck - ***Test Driven Development: By Example***. Addison-Wesley Longman, 2002.
2. Kleinberg and Tardos – ***Algorithm Design***. Pearson Educational, 2014 (http://www.cs.princeton.edu/~wayne/kleinberg-tardos/)
3. Martin Fowler - ***Refactoring. Improving the Design of Existing Code***. Addison-Wesley, 1999. (http://refactoring.com/catalog/index.html)
4. Frentiu, M., H.F. Pop, Serban G. - ***Programming Fundamentals***, Cluj University Press, 2006
5. ***The Python language reference***. (https://docs.python.org/3/reference/index.html)
6. ***The Python standard library***. (https://docs.python.org/3/library/index.html)
7. ***The Python tutorial***. (https://docs.python.org/3/tutorial/index.html)


| **8.2 Seminar** | Teaching Methods | Remarks |
|---|---|---|
| 1. Introduction to Python. Simple problems | • Interactive exposure | The seminar is |
| 2. Procedural Programming | | |

| | | |
|---|---|---|
| 3. Modular Programming (I)<br>4. Modular Programming (II)<br>5. Object Based Programming<br>6. User Defined Types<br>7. Program Design (I). Layered Architecture<br>8. Program Design (II). Layered Architecture<br>9. Program Design (III). Inspection and Testing<br>10. Recursion. Algorithm Complexity<br>11. Searching. Sorting.<br>12. Program Design Recap<br>13. Problem Solving Methods: Greedy, Divide & Conquer<br>14. Problem Solving Methods: Backtracking, Dynamic Programming | • Explanation<br>• Conversation<br>• Didactical demonstration | structured as a weekly 2 hour class. |

**Bibliography**


1. Kent Beck - ***Test Driven Development: By Example***. Addison-Wesley Longman, 2002.
2. Kleinberg and Tardos – ***Algorithm Design***. Pearson Educational, 2014 (http://www.cs.princeton.edu/~wayne/kleinberg-tardos/)
3. Martin Fowler - ***Refactoring. Improving the Design of Existing Code***. Addison-Wesley, 1999. (http://refactoring.com/catalog/index.html)
4. Frentiu, M., H.F. Pop, Serban G. - ***Programming Fundamentals***, Cluj University Press, 2006
5. ***The Python language reference***. (https://docs.python.org/3/reference/index.html)
6. ***The Python standard library***. (https://docs.python.org/3/library/index.html)
7. ***The Python tutorial***. (https://docs.python.org/3/tutorial/index.html)


| **8.3 Laboratory** | Teaching Methods | Remarks |
|---|---|---|
| 1. Simple Python program<br>2. Feature-driven software development process (I)<br>3. Feature-driven software development process (II)<br>4. Feature-driven software development process (III)<br>5. Laboratory test<br>6. Layered architecture (I)<br>7. Layered architecture (II)<br>8. Layered architecture (III)<br>9. Text Files<br>10. Program Testing<br>11. Algorithm Complexity<br>12. Problem Solving Methods<br>13. Laboratory test – practical exam simulation<br>14. Assignment delivery time | • Explanation<br>• Conversation | • The lab is structured as weekly 2 hour classes.<br>• Laboratory assignments are due 1 week after assignment. |

**Bibliography**


1. Kent Beck - ***Test Driven Development: By Example***. Addison-Wesley Longman, 2002.
2. Kleinberg and Tardos – ***Algorithm Design***. Pearson Educational, 2014 (http://www.cs.princeton.edu/~wayne/kleinberg-tardos/)
3. Martin Fowler - ***Refactoring. Improving the Design of Existing Code***. Addison-Wesley, 1999. (http://refactoring.com/catalog/index.html)
4. Frentiu, M., H.F. Pop, Serban G. - ***Programming Fundamentals***, Cluj University Press, 2006
5. ***The Python language reference***. (https://docs.python.org/3/reference/index.html)
6. ***The Python standard library***. (https://docs.python.org/3/library/index.html)

7. ***The Python tutorial***. (https://docs.python.org/3/tutorial/index.html)

**9. Corroborating the content of the discipline with the expectations of the epistemic community, professional associations and representative employers within the field of the program.**

The course respects the IEEE and ACM Curricula Recommendations for Computer Science studies.
The course exists in the studying program of all major universities in Romania and abroad.
The content of the course is considered the software companies as important for average programming skills

**10. Evaluation**

| Type of activity | 10.1 Evaluation Criteria | 10.2 Evaluation Methods | 10.3 Share in the grade (%) |
|---|---|---|---|
| 10.4 Lecture | The correctness and completeness of the accumulated knowledge and the capacity to design and implement correct Python programs | Written exam (during the regular session) | **40%** |
| 10.5 Seminar/ Laboratory | Be able to design, test and debug a Python program | Practical evaluation (in the regular session) | **30%** |
| | Correctness of delivered laboratory assignments and documentation | Program and documentation portfolio | **30%** |
| 10.6 Minimum performance standards | | | |

- Students must observe the standards of academic integrity.
- Each student has to prove that they acquired an acceptable level of knowledge and understanding of the core concepts taught in the class, that they are capable of using knowledge in a coherent form, that they have the ability to establish certain connections and to use the knowledge in solving different problems in programming.
- Successfully passing the exam is conditioned by a minimum grade of 5 at the lab activity, practical test and written examination.

| Date | Signature of course coordinator | Signature of seminar coordinator |
|---|---|---|
| 20.04.2018 | Lect. PhD. Molnar Arthur | Lect. PhD. Molnar Arthur |

Date of approval

Signature of the head of department

Prof. PhD. Anca Andreica